CONSENSYS
Diligence

ChainBridge

Date	May 2020
Lead Auditor	Sergii Kravchenko
Co-auditors	Daniel Luca

1 Executive Summary

This report presents the results of our engagement with ChainSafe Systems to review ChainBridge, a bridging solution between EVM-compatible blockchains and Substrate-based chains, at the time of writing.

The review was conducted over the course of three and a half calendar weeks, from April 22 to May 15, 2020 by Sergii Kravchenko and Daniel Luca. A total of 36 person-days were spent.

Following our initial review of the ChainBridge codebase, the ChainSafe team implemented changes to address our findings. A total of 40 hours were spent to review these changes.

1.1 Scope

Our review focused on the three repositories listed below, each with its corresponding commit hash:

- https://github.com/ChainSafe/ChainBridge-solidity
  - Commit hash: `03272f6bb4756af2fd5ee5c89f9e70c297833cb3`
- https://github.com/ChainSafe/ChainBridge
  - Commit hash: `a28daab880ccc653a790deeca40be12769618781`
- https://github.com/ChainSafe/chainbridge-substrate
  - Commit hash: `48e48ae579ab740a73dd5a702e3b5f5d88f245c6`

ChainSafe's stated objective was to understand the current condition of their system, as well as what changes are needed in order to improve the system's security. The intended audience of the report is the ChainSafe team and their clients.

Note: in the Substrate codebase, only the main library is in scope. Example pallets are excluded from the review.

2 Key Observations

- The on-chain code for different blockchains (Solidity contracts for EVM, and Rust runtime modules in Substrate) has slightly different functionality:
  - Within the Substrate component, relayers can vote against the proposals, which is not an option in Solidity code.
  - There are some differences in different blockchains due to their heterogeneous core principles. (e.g., transaction cost distribution)
- A significant amount of configuration and additional manual work, such as creating and managing new token contracts/pallets, is required for the whole system to operate properly,
- There are no security checks in inter-pallet communication on the Substrate side. For example, any pallet can potentially call the `transfer_fungible` function in the *Bridge* pallet and then mint/release tokens on the other blockchain. It is therefore essential to ensure that every module in the runtime is safe. The reason for this is that Substrate does not provide an easy way to define where the function is called from.

The class of issues arising from the nuance outlined in the text above make it easier to miss important details. Pallets may have arbitrary call opportunities inside them that expand the attack surface. These attack vectors are often related to the difficulty of validating the input space.

On-chain system components

The on-chain component of the system can be split into two main functions, each of which has a few components:

1. **Transferring out**:
   1. *Deposit processing* - handling deposits of different kinds (fungible, non-fungible tokens, generic calls).
   2. *Bridge* - emitting event.
2. **Transferring in**:
   1. *Transfer execution* - transferring tokens or calling generic functions.
   2. *Bridge* - voting for proposals.

Additionally, the *Bridge* component is also used for configuration and administration actions, including editing the relayers list, resources, chains, and more.

Each of these main functions should be implemented in both Solidity and Substrate. However, the Substrate component we revieved only includes the *Bridge* implementation. Both *Deposit processing* and *Transfer execution* have yet to be implemented.

3 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify many of the specific security properties that were the focus of our review.

3.1 Actors

The relevant actors are listed below with their respective abilities:

- Bridge Admin
  - Can add or remove `Relayers`
  - Can change the number of votes for a proposal to pass
  - Can pause and unpause the `Bridge`
  - Can transfer funds in the contract to any account. (EVM-specific)
  - Register/whitelist resource and chain IDs.
- Relayer
  - Can vote or create new `Proposals`
  - Can execute passed `Proposals`
  - Can vote against `Proposals` (Substrate-specific)
- User
  - Can transfer tokens between blockchains by locking tokens in `Bridges`.

3.2 Trust Model

In the distributed systems being analyzed, it is essential to identify what trust is expected/required between various actors. For this assessment, we established the following trust model:

- The system is completely trusted; admin users have complete control of all components reviewed. For example, minting an infinite amount of tokens or withdrawing all the tokens from the Bridge contract on the original chain.
- The system assumes that there is no `_relayerThreshold` number of malicious relayers, or else they may steal funds from the contracts.
- Users should have full trust in admin and relayers-based consensus not only when using the Bridge, but also when using tokens created by the Bridge (tokens on a foreign blockchain that represent the original tokens).

4 Issues

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

4.1 Any relayer can replace the proposal to drain funds Critical ✓ Addressed

Resolution

Addressed in ChainSafe/chainbridge-solidity#146. The code now checks that the `datahash` is valid. However, there is still an opportunity for a malicious relayer to register an invalid `datahash` first. In this case, other relayers will not be able to register and vote for the valid one, and the transfer will not be processed until the admin processes it.

Update: The new issue is addressed in ChainSafe/chainbridge-solidity#184. Now, the `datahash` parameter is a part of the key for proposals, so one malicious relayer cannot prevent the others from voting for a valid proposal.

Description

Relayers can see any incoming transaction from other actors in the system. Especially important for the vulnerability described below, is the case where a *non-relayer* actor deposits tokens on blockchain A to transfer them to blockchain B (EVM-based).

In addition to the above, another essential condition that enables the exploitation of the vulnerability presented below, is the fact that relayers can all call the `voteProposal` function with defined `depositNonce` and `dataHash`, as noticeable by the line depicted below.

contracts/Bridge.sol:L294

```
function voteProposal(uint8 chainID, uint256 depositNonce, bytes32 dataHash) public
onlyRelayers whenNotPaused {
```

As we can observe in the logic of function mentioned above, if a proposal has already been submitted a relayer will vote for it, if not, a new proposal is created.

The exploitable issue arises from the fact that there is no verification that `dataHash` is the same as in an existing proposal. Effectively allowing any relayer to be the first one to create the proposal (possibly even doing it beforehand) and "deceiving" every other relayer into automatically voting for it.

This means that any relayer can replace the intended "benign" proposal with a malicious proposal that withdraws all the tokens from the smart contract.

Substrate note: `datahash` is already a part of the key in the Substrate code. Therefore, this specific issue is not present in Substrate-based chains.

Recommendation

Ensure that the `dataHash` of the proposal matches the `dataHash` parameter that is passed in the `voteProposal` function.

4.2 Transfer functions of ERC20 tokens may return `false` Critical ✓Addressed

Resolution
Addressed in https://github.com/ChainSafe/chainbridge-solidity/pull/152/ by adding a `_safeTransfer` function to all transfers. Additionally, the same functionality was added to ERC-721 token transfers even though they do not return anything by the standard.

Description

ERC20 token transfers are used in multiple functions of the `ERC20Safe` contract:

contracts/ERC20Safe.sol:L29-L34

```
function fundERC20(address tokenAddress, address owner, uint256 amount) public {
    IERC20 erc20 = IERC20(tokenAddress);
    erc20.transferFrom(owner, address(this), amount);

    _balances[tokenAddress] = _balances[tokenAddress].add(amount);
}
```

contracts/ERC20Safe.sol:L44-L49

```
function lockERC20(address tokenAddress, address owner, address recipient, uint256
amount) internal {
    IERC20 erc20 = IERC20(tokenAddress);
    erc20.transferFrom(owner, recipient, amount);

    _balances[tokenAddress] = _balances[tokenAddress].add(amount);
}
```

contracts/ERC20Safe.sol:L58-L63

```
function releaseERC20(address tokenAddress, address recipient, uint256 amount)
internal {
    IERC20 erc20 = IERC20(tokenAddress);
    erc20.transfer(recipient, amount);

    _balances[tokenAddress] = _balances[tokenAddress].sub(amount);
}
```

These `transfer` and `transferFrom` calls may just return `False` instead of canceling the whole current transaction.

Substrate note: This part of the logic is absent in the substrate library and should still be written in the actual implementation.

Recommendation

Properly process ERC20 transfers.

4.3 Re-entrancy attack on `executeProposal` function Critical ✓ Addressed

Resolution

Addressed in ChainSafe/chainbridge-solidity#151. `proposal.status` is now set before the transfer. Note that no reentrancy protection was added.

Description

When relayers have already voted for a proposal, any relayer can execute it by calling `executeProposal`.

contracts/Bridge.sol:L336-L349

```
function executeProposal(uint8 chainID, uint256 depositNonce, address handler, bytes
memory data) public onlyRelayers whenNotPaused {
    Proposal storage proposal = _proposals[uint8(chainID)][depositNonce];

    require(proposal._status != ProposalStatus.Inactive, "proposal is not active");
    require(proposal._status == ProposalStatus.Passed, "proposal was not passed or
has already been transferred");
    require(keccak256(abi.encodePacked(handler, data)) == proposal._dataHash,
        "provided data does not match proposal's data hash");

    IDepositExecute depositHandler = IDepositExecute(handler);
    depositHandler.executeDeposit(data);

    proposal._status = ProposalStatus.Transferred;
    emit ProposalExecuted(chainID, _chainID, depositNonce);
}
```

Meaning that, upon a token transfer, which happens in the following line:

`depositHandler.executeDeposit(data);`

it is possible to exploit a reentrancy opportunity in permitting tokens (e.g., some implementations of the ERC777 standard), and call `executeProposal` again.

The exploitation presented above works because the status of the proposal is changed after the transfer:

`proposal._status = ProposalStatus.Transferred;`

Substrate note: This part of the logic is absent in the Substrate library and should still be written in the actual implementation. When it is implemented, some version of “reentrancy” can, potentially, be an issue.

Recommendation

The two possible solutions for the vulnerability presented are:

1. `proposal._status = ProposalStatus.Transferred;` should be called before the token transfer.
2. Add reentrancy protection.

4.4 Any function can be called during generic deposit Major ✓ Addressed

Resolution

Addressed in ChainSafe/chainbridge-solidity#104 and ChainSafe/chainbridge-solidity#115. Now there are separate, predefined functions that are called during the `deposit` and the `executeDeposit` processes.

Description

When the `deposit` function of the generic handler is called, the `GenericHandler` contract should call a specific method of a specific contract, that is bound to `resourceID`.

contracts/handlers/GenericHandler.sol:L177-L183

```
address contractAddress = _resourceIDToContractAddress[resourceID];
require(_contractWhitelist[contractAddress], "provided contractAddress is not
whitelisted");

if (_contractAddressToDepositFunctionSignature[contractAddress] != bytes4(0)) {
    (bool success,) = contractAddress.call(metadata);
    require(success, "delegatecall to contractAddress failed");
}
```

The issue arises from the fact that the user can specify a different function in the `metadata` parameter, effectively calling this “malicious” function instead of the originally intended one.

Substrate note: This part of the logic is absent in the Substrate library and should still be written in the actual implementation.

Recommendation

Check if `metadata` contains the same function as `_contractAddressToDepositFunctionSignature[contractAddress]`.

4.5 Possible incorrect registration of events in case of chain re-orgs Major ✓ Addressed

Resolution

Addressed in ChainSafe/ChainBridge#410. The Ethereum listener now waits for 10 blocks before processing. The Substrate listener now only processes finalized blocks.

Description

As soon as a new block is included in a chain, events are processed and actions are executed. This creates the risk of processing some events that are part of a block reorg.

chains/ethereum/listener.go:L101-L105

```
// Sleep if the current block > latest
if currBlock.Number.Cmp(latestBlock) == -1 {
        time.Sleep(BlockRetryInterval)
        continue
}
```

Examples

Reorgs happen quite often on Ethereum. You can see the uncle rate in the link below.

https://etherscan.io/blocks_forked

> ***Substrate note:*** *Substrate chains might have different consensus algorithms in its configuration. Depending on the consensus mechanism there should be a different number of confirmation blocks (`0` when an algorithm with instantaneous finality is used).*

Recommendation

Enforce a delay of at least 10 blocks. This will reduce the number of processed uncles to a number close to zero.

4.6 Voting result is not checked properly Medium ✓Addressed

Resolution

Addressed in ChainSafe/chainbridge-substrate#56.

Description

On the Substrate side, during voting periods, the following function is called to define whether the vote is completed or not:

chainbridge/src/lib.rs:L60-L70

```
fn try_to_complete(&mut self, threshold: u32, total: u32) -> bool {
    if self.votes_for.len() >= threshold as usize {
        self.status = ProposalStatus::Approved;
        true
    } else if self.votes_against.len() > (total - threshold) as usize {
        self.status = ProposalStatus::Rejected;
        true
    } else {
        false
    }
}
```

However, after the call is completed, the result of the vote is not properly checked. Only the fact that it has finished is taken into account.

While in most cases it is true that the result of the completed vote can be predicted, there are some edge-cases when this is not true. For example, when the `total` or `threshold` variables are updated. This may lead to executing proposals that should be canceled and vice versa.

> ***Ethereum note:*** *the Solidity implementation does not allow voting against a proposal, hence no* `cancel_execution` *is possible. So, when a vote is complete, the result can only be positive.*

Recommendation

Check the result of the vote after it is completed.

4.7 Error handling in Substrate Medium ✓Addressed

Resolution

- This issue's first example is invalid because all the checks in `<bridge::Module<T>>::transfer_fungible` are duplicated here https://github.com/ChainSafe/chainbridge-substrate/blob/48e48ae579ab740a73dd5a702e3b5f5d88f245c6/example-pallet/src/lib.rs#L74, ensuring that this call to the Bridge will not fail.
- In the second example, if the execution of a proposal fails, it will still be considered as executed. That is the intended behavior; if the proposal fails that way, there is a manual mechanism for admin to return funds on the Ethereum side.

The issue severity is downgraded from "Major" to "Medium" after getting the comments from the client.

Description

It is much harder to handle any error in Substrate than in Ethereum smart contracts. While on Ethereum one can use the `revert` opcode and be sure that all storage reverts to the previous state, the same is not true for Substrate. All storage changes that occur before an error is thrown remain intact.

The best practice in Substrate is to follow the "verify first, write last" principle. During execution, one should guarantee all the checks are using the `ensure` macro and that, upon starting to modify storage, no error can be thrown after.

While the outlined approach sounds clean, it can get very complicated when inter-pallet communication is involved. It is especially true when multiple different external pallets need to be called in one function. In such cases, it becomes extremely difficult to perform all the verifications before changing storage.

Examples

There are a few places where this problem may occur:

- `transfer_fungible`, `transfer_nonfungible`, `transfer_generic` functions of the Substrate library are called during the deposit from another pallet. They make use of the `ensure` macro, so they may potentially fail.

chainbridge/src/lib.rs:L487-L506

```
    pub fn transfer_fungible(
        dest_id: ChainId,
        resource_id: ResourceId,
        to: Vec<u8>,
        amount: U256,
    ) -> DispatchResult {
        ensure!(
            Self::chain_whitelisted(dest_id),
            Error::<T>::ChainNotWhitelisted
        );
```

```
        let nonce = Self::bump_nonce(dest_id);
        Self::deposit_event(RawEvent::FungibleTransfer(
            dest_id,
            nonce,
            resource_id,
            amount,
            to,
        ));
        Ok(())
    }
```

This means that these functions should be called either before any storage changes (outside of current pallet) or the same checks should be done in the outside pallet, before changing the state. Even though example pallets are out of scope, in this example, you can see that the actual transfer is happening before calling the `transfer_fungible` function:

example-pallet/src/lib.rs:L72-L80

```
    pub fn transfer_native(origin, amount: BalanceOf<T>, recipient: Vec<u8>, dest_id:
bridge::ChainId) -> DispatchResult {
        let source = ensure_signed(origin)?;
        ensure!(<bridge::Module<T>>::chain_whitelisted(dest_id), Error::
<T>::InvalidTransfer);
        let bridge_id = <bridge::Module<T>>::account_id();
        T::Currency::transfer(&source, &bridge_id, amount.into(), AllowDeath)?;

        let resource_id = T::NativeTokenId::get();
        <bridge::Module<T>>::transfer_fungible(dest_id, resource_id, recipient,
U256::from(amount.saturated_into()))
    }
```

That may lead to tokens actually being transferred to the Bridge, but the event to transfer them outside is not emitted.

- When a proposal passes, its state is saved to storage before calling `finalize_execution` function.

chainbridge/src/lib.rs:L421-L429

```
let complete = votes.try_to_complete(<RelayerThreshold>::get(),
<RelayerCount>::get());
<Votes<T>>::insert(src_id, (nonce, prop.clone()), votes.clone());
Self::deposit_event(RawEvent::VoteFor(src_id, nonce, who.clone()));

if complete {
    Self::finalize_execution(src_id, nonce, prop)
} else {
    Ok(())
}
```

inside of the `finalize_execution` function, an external pallet is called (whose implementation is still inexistent at the time of writing), which can also potentially fail. Moreover, even in the event the scenario described above materializes, the proposal will still be considered as executed.

Recommendation

Make sure no errors are being thrown after storage changes or build a mechanism that reverts all changes after an error is thrown.

4.8 Should check length before slicing byte arrays when handling events Medium ✓ Addressed

Resolution

Addressed in ChainSafe/ChainBridge#405. The ChainBridge developers provided the following comment:

> *This check is actually unnecessary. The length is required in the solidity assembly, but the listener will receive bytes of the correct length when querying the deposit record.*

Description

When an ERC 721 Deposit is handled, the recipient's address is retrieved from the event.

chains/ethereum/events.go:L34-L39

```
record, err := l.erc721HandlerContract.GetDepositRecord(&bind.CallOpts{},
nonce.Big(), uint8(destId))
if err != nil {
        l.log.Error("Error Unpacking ERC20 Deposit Record", "err", err)
}

recipient :=
record.DestinationRecipientAddress[:record.LenDestinationRecipientAddress.Int64()]
```

The length of the address is handled in the ERC 721 handler.

contracts/handlers/ERC721Handler.sol:L117-L131

```
// Load length of recipient address from data + 96
lenDestinationRecipientAddress := mload(add(data, 0x60))
// Load free mem pointer for recipient
destinationRecipientAddress := mload(0x40)
// Store recipient address
mstore(0x40, add(0x20, add(destinationRecipientAddress,
lenDestinationRecipientAddress)))

// func sig (4) + destinationChainId (padded to 32) + depositNonce (32) + depositor
(32) +
// bytes lenght (32) + resourceId (32) + tokenId (32) + length (32) = 0xE4

calldatacopy(
    destinationRecipientAddress,    // copy to destinationRecipientAddress
    0xE4,                           // copy from calldata after
destinationRecipientAddress length declaration @0xE4
    sub(calldatasize(), 0xE4)       // copy size (calldatasize - (0xE4 + 0x20))
)
```

However, token handlers can be swapped and the event handler should not rely on the correctness and safeness of the token handler.

The problem is that if the handler generates an event with a `DestinationRecipientAddress` shorter than `LenDestinationRecipientAddress`, the node will panic.

Recommendation

The node should be able to return an error, in case the decoding is faulty.

Such a case should be when `LenDestinationRecipientAddress > len(DestinationRecipientAddress)`.

4.9 Anyone can pass any `handler` in deposit Medium ✓ Addressed

Resolution
Addressed in ChainSafe/chainbridge-solidity#131. The handler's address is now bound to the `resourceID` and is set by the admin.

Description

Anyone can call the `deposit` function to transfer funds from a supported chain to any other.

contracts/Bridge.sol:L269-L279

```
function deposit (uint8 destinationChainID, address handler, bytes memory data)
public payable whenNotPaused {
    require(msg.value == _fee, "Incorrect fee supplied");

    uint256 depositNonce = ++_depositCounts[destinationChainID];
    _depositRecords[destinationChainID][depositNonce] = data;

    IDepositExecute depositHandler = IDepositExecute(handler);
    depositHandler.deposit(destinationChainID, depositNonce, msg.sender, data);

    emit Deposit(destinationChainID, handler, depositNonce);
}
```

The issue arises from the fact that the caller can pass any address as the `handler`. In addition, the `Bridge` contract will call this address, which might prove unsafe under certain conditions.

> ***Substrate note:*** *This part of the logic is absent in the Substrate library and should still be written in the actual implementation.*

Recommendation

Only allow passing an authorized `handler` address as the parameter.

4.10 `_balances` does not represent contract balance as it should

Medium **✓ Addressed**

Resolution
Addressed in ChainSafe/chainbridge-solidity#149.

Description

In both the `ERC20Safe` and `ERC721Safe` contracts, the `_balances` field should represent the current balance of the contract. During minting, `_balances` is increased even though the recipient of the minting is not the contract itself but an external agent:

contracts/ERC20Safe.sol:L72-L77

```
function mintERC20(address tokenAddress, address recipient, uint256 amount) internal
{
    ERC20PresetMinterPauser erc20 = ERC20PresetMinterPauser(tokenAddress);
    erc20.mint(recipient, amount);

    _balances[tokenAddress] = _balances[tokenAddress].add(amount);
}
```

contracts/ERC721Safe.sol:L73-L78

```
function mintERC721(address tokenAddress, address recipient, uint256 tokenID, bytes
memory data) internal {
    ERC721MinterBurnerPauser erc721 = ERC721MinterBurnerPauser(tokenAddress);
    erc721.mint(recipient, tokenID, string(data));

    _balances[tokenAddress] = _balances[tokenAddress].add(1);
}
```

> ***Substrate note:*** *This part of the logic is absent in the substrate library and should still be written in the actual implementation.*

Recommendation

Do not increase `_balances` during minting if the recipient is not the contract itself. Or, in alternative, use the `balanceOf` function of the token, instead.

4.11 Total relayers count may be wrong Medium ✓ Addressed

Resolution
Addressed in ChainSafe/chainbridge-solidity#147.

Description

If the admin adds a relayer that was previously admitted, already:

contracts/Bridge.sol:L161-L165

```
function adminAddRelayer(address relayerAddress) public onlyAdmin {
    grantRole(RELAYER_ROLE, relayerAddress);
    emit RelayerAdded(relayerAddress);
    _totalRelayers++;
}
```

or removes a relayer that was already removed:

contracts/Bridge.sol:L173-L177

```
function adminRemoveRelayer(address relayerAddress) public onlyAdmin {
    revokeRole(RELAYER_ROLE, relayerAddress);
    emit RelayerRemoved(relayerAddress);
    _totalRelayers--;
}
```

`_totalRelayers` will still change by an absolute value of 1 even though the effective number of relayers is left unchanged. In addition, the variable can underflow/overflow.

> ***Substrate note:*** *the Substrate library is already implementing the proper checks when adding/removing relayers.*

Recommendation

Make sure that admin is adding/removing non-existing/existing relayer.

4.12 Don’t route event if handler is not valid Medium ✓Addressed

Resolution

Addressed in ChainSafe/ChainBridge#407. The function now returns on error. As a result, all events left in the block after the failed message will not be processed.

Update: An unrecognized handler is now skipped, rather than throwing an error ChainSafe/ChainBridge#458.

Description

An event should never be sent to the router if it does not match any of the recognized handlers.

An error is logged, but the execution is not stopped.

chains/ethereum/listener.go:L141-L154

```
if addr == l.cfg.erc20HandlerContract {
        m = l.handleErc20DepositedEvent(destId, nonce)
} else if addr == l.cfg.erc721HandlerContract {
        m = l.handleErc721DepositedEvent(destId, nonce)
} else if addr == l.cfg.genericHandlerContract {
        m = l.handleGenericDepositedEvent(destId, nonce)
} else {
        l.log.Error("Event has unrecognized handler", "handler", addr)
}

err = l.router.Send(m)
if err != nil {
        l.log.Error("subscription error: failed to route message", "err", err)
}
```

Recommendation

Skip routing the event if the handler does not match the allowed ones.

4.13 Proposal voting can only be completed when making a vote

Minor ✓ Addressed

Resolution

Addressed in ChainSafe/chainbridge-substrate#57. Now voting is separated from execution.

Description

In Substrate, it's only possible to calculate voting results when a new vote is submitted.

However, if `total` or `threshold` are updated, it should be possible to check the results of the current votings and call `finalize_execution` (or `cancel_execution`) without submitting a vote. This very closely resembles a similar problem presented in the Solidity implementation.

Even though it is possible to execute the proposal without making a vote, the proposal's status should be `ProposalStatus.Passed`, which can only be assigned when making a vote.

Recommendation

Add a public function that checks if a vote can be completed and executed.

4.14 Overflow/underflow handling in Substrate Minor ✓ Addressed

Resolution

Addressed in ChainSafe/chainbridge-substrate#58. If `total < threshold`, a proposal cannot be rejected.

Description

Substrate does not have overflow/underflow protection by default. In order to avoid this class of bugs, it is recommended to use safe functions like `checked_add()` and `checked_sub()`. In the current codebase math operations are mostly either incrementing (`+1`) or making operations with values that shouldn't overflow/underflow logically:

chainbridge/src/lib.rs:L64

```
} else if self.votes_against.len() > (total - threshold) as usize {
```

Even though, in this specific instance, this does not present a big risk, if the `threshold` is bigger than `total`, an underflow will occur.

> ***Ethereum note:*** *SafeMath library is already used Ethereum contracts for most of the math operations.*

Recommendation

Use safe functions like `checked_add()` and `checked_sub()` for math operations.

4.15 `setResource` can be simplified Minor ✓Addressed

Resolution
Addressed in ChainSafe/chainbridge-solidity#153.

Description

contracts/handlers/HandlerHelpers.sol:L42-L45

```
bytes32 currentResourceID = _tokenContractAddressToResourceID[contractAddress];
bytes32 emptyBytes;
require(keccak256(abi.encodePacked((currentResourceID))) ==
```

```
    keccak256(abi.encodePacked((emptyBytes))),
        "contract address already has corresponding resourceID");
```

A comparison of `bytes32` to zero can be done directly (`currentResourceID == bytes32(0)`) without using hash functions.

4.16 Proposals cannot be cancelled Minor ✓ Addressed

Resolution

Addressed in ChainSafe/chainbridge-solidity#156 and ChainSafe/chainbridge-substrate#59. Now, proposals expire after some time. The Ethereum side makes use of a centralized mechanism for returning funds. On the Substrate side, this mechanism has yet to be implemented.

Description

If there are not enough votes for a proposal to pass, that same proposal will be kept in the system forever and can, therefore, be voted on and executed later. The issue arises from the fact that it is impossible to cancel a proposal and be sure that it will never be executed, when you want to send a newer, updated one at a later point in time.

This issue applies to both EVM-compatible and Substrate-based chains.

Recommendation

A possible, and recommended, way to mitigate the issue is to introduce logic for proposals to have deadlines.

Appendix 1 - Activity log

A.1.1 Preparing for the review

Prior to the review, we were introduced to the ChainBridge system and the different technologies within it. We took a few days before the engagement to get more familiar with the intricacies of the Rust language.

During our kick-off call with ChainSafe, we discussed their objectives, confirmed the precise scope of the review, and identified the review priority for each of the system's components. The remainder of the call was spent walking through the system's purpose and different components.

We spent time going through the provided documentation and split our efforts going through each repository.

A.1.2 Week 1

During the first week of the engagement, we used the documentation to try setting up the system locally to test different attack vectors. A few obsolete documentation sections were identified, and the client was notified.

On April 27, 2020, we had another code walkthrough with the client going through all three repositories, focusing mostly on how the components work together.

On April 30, 2020, we received the final commit hashes for each repository; they are included in the Scope section.

We spent the remainder of the week focused on different repositories. However, the focus was not exclusive; each member checking code from different repositories to better understand how the components interact with each other.

A.1.3 Week 2

The second week was used to further our understanding of the system while filing issues we found along the way.

On May 6, 2020, we had a code walkthrough, this time focusing on the Golang repository. A few possible issues were discussed during the call with the client.

On May 8, 2020, we scheduled another code walkthrough for the Rust component of the system.

A.1.4 Week 3

The third week was used to review the Rust repository while also filing issues for the rest of the system.

Towards the end of the week, we began drafting a report of the engagement.

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.